Optical content-addressable parallel processor: architecture, algorithms, and design concepts

Ahmed Louri

Associative processing based on content-addressable memories has been argued to be the natural solution for nonnumerical information processing applications. Unfortunately, the implementation requirements of these architectures when one uses conventional electronic technology have been cost prohibitive; therefore associative processors have not been realized. Instead, software methods that emulate the behavior of associative processing have been promoted and mapped onto conventional location-addressable systems. However, this does not bring about the natural parallelism of associative processing, namely, the ability to access many data words simultaneously. Optics has the advantage over electronics of directly supporting associative processing by providing economic and efficient interconnects, massive parallelism, and high-speed processing. The principles of designing an optical content-addressable parallel processor (OCAPP) for the efficient support of parallel symbolic computing are presented. The architecture is designed to exploit optics advantages fully in interconnects and high-speed operations. Several parallel search-and-retrieval algorithms are mapped onto an OCAPP to illustrate its capability of supporting parallel symbolic computing. A theoretical performance analysis of these algorithms is presented. This analysis reveals that the execution times of the parallel algorithms presented are independent of the problem size, which makes the OCAPP suitable for applications in which the number of data sets to be operated on is high (e.g., massive parallel processing). A preliminary optical implementation of the architecture with currently available optical components is also presented.

I. Introduction

The information explosion seen in recent years in all fields of human endeavor has stimulated the development of computer-based information systems to assist in the creation, storage, modification, classification, and retrieval of mainly textual or symbolic data. For example, progress in database management systems, expert systems, and intelligent knowledge-based systems is increasing demand for symbolic information processing such as text editing, file processing, table sorting, searching, and retrieval, which have no numerical meaning. In fact, a substantial proportion of the work load of modern information processing systems involves searching and sorting symbolic data.[1,2] Nevertheless, a majority of computers are designed mainly for numerical computations, and they suffer from a fundamental handicap that stems from the principle of addressing the memory. The notion of locating information by its address is fundamentally a weak design concept for symbolic data processing.

When a search for a value is made through a location-addressable memory, the entire memory may need to be searched one word at a time (if the data are not sorted in memory), which consumes a great deal of time. There is no logical reason why the search must be done sequentially. The only reason stems from the fundamental handicap of separating processing and memory and addressing memory one word at a time. This fundamental flaw has forced system analysts and programmers to develop sophisticated software techniques for symbolic information processing such as hashing and indexing.[3,4] However, the implementation of such software techniques on location-addressed computers has lead to complex, expensive, and inefficient information processing systems.[5,3]

Searching, retrieving, sorting, and modifying symbolic data can be significantly improved by the use of

The author is with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona 85721.

Received 16 November 1990.
0003-6935/92/173241-18$05.00/0.

content-addressable memory instead of location addressability.[3,6] In such a method, the absolute location of each data object has no logical significance; all access to data objects is by content. As an illustration, consider a professor trying to find out how many students in his class are familiar with the optical computing subject. If he considers the students to be a coordinate-addressed memory, he asks the questions: Does the student in row one, column one know the subject? Does the student in row one, column two know the subject? He does this one at a time until he exhausts all the seats in the classroom. If the professor assumes a content-addressable memory, he stands before the class and says: If you have prior knowledge of optical computing, please raise your hand. He then gets multiple responses (if any) at the same time. In concept, such associative processing is a naturally parallel form of symbolic representation and manipulation of abstract data structures, and has potential benefits in simplicity of expression (programming), storage capacity, and speed of execution.[3,7,8] There are two hypotheses underlying this paper:

(1) Associative processing provides a sound basis to uncover inherent parallelism in symbolic processing and information retrieval systems.

(2) Optics is potentially the ideal medium to exploit such parallelism by providing efficient implementation support for the associative processing model.

The rest of the paper is organized as follows. Section II briefly reviews associative processing. This includes a basic architecture, benefits, and difficulties faced by electronics for implementing this computing model. Section III presents the design concepts and structural organization of an optical content-addressable parallel processor (OCAPP). Section IV describes a variety of parallel algorithms for searching and information retrieval that can be efficiently implemented on an OCAPP, and in Section V we consider the optical implementation of the different components of an OCAPP. Here, only a general description of the implementation issues will be considered. Section VI presents an estimated performance analysis, and Section VII concludes the paper.

II Associative Processing

A. Background

In an associative memory, information is addressed by its contents.[3] An associative processor is a parallel processing machine in which the data items are content addressable, and it has the added capability of writing in parallel into words satisfying certain criteria. It may be that the entire contents of stored words may be changed, or just a few bits of the words. A basic associative architecture that can provide a parallel search and update is depicted in Fig. 1.[3] It consists of n words of associative memory, word select

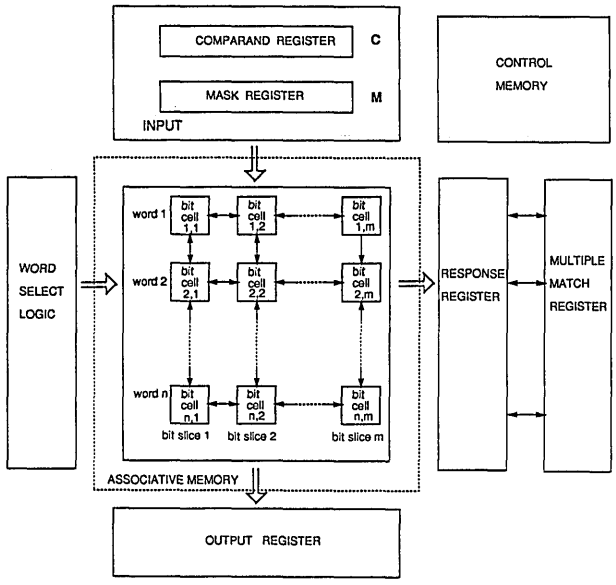

Fig. 1. Basic architecture of an associative memory processor.

logic, bit-slice select logic, a control memory for storing programs and control, response register, multiple match resolver, an output register, and some auxiliary circuits for control. The data memory consists of n words with m bits per word. Each bit cell in the memory contains storage and comparison logic to determine whether it matches an interrogating signal broadcast by the bit-slice select logic, and also for parallel read and white. In the bit-slice select logic, the comparand register C is used to hold the key operand that is being searched for. The mask register M is used to enable or disable the bit slices involved in the parallel comparison operations across all the words of the memory array. The response register R is used to indicate if there are matching words. The multiple response resolver is used to select one (it may be the first) of the matching words. Parallel comparisons are performed on all memory words.

This memory organization leads to a computational model with the following advantages over conventional location-addressable models[9,10]:

- Information is processed within the associative memory, without transfer to an independent processing unit. Since there is no addressing of data and no data movement, this implies the elimination of the fundamental von Neumann bottleneck encountered in conventional systems.
- The amount of time required for searching, retrieving, and updating information is independent of the memory and the problem sizes (e.g., the number of words involved).
- There is no restriction on the length of words that can be stored in the memory. This implies a computational environment with a variable word length, as opposed to conventional systems in which the word length is fixed at design time.

- Because of the modular and uniform nature of the associative memory model, an associative processor can be easily expanded by adding identical cells and by adjusting the control registers accordingly without any effect on the software and the control design. This implies great potential for scalability and expandability.

B. Problems with Electronic Content-Addressable Memories

Despite the above advantages of associative processing, this model of computing is not extensively used because of the difficulty and high cost of implementing it in conventional electronic technology.[5,3] Instead, software methods such as hashing and indexing that emulate a content-addressable capability have been used. The major problems with current electronic content-addressable memories (CAM's) are the following:

(1) Each bit cell in an associative memory is much more complex and requires more circuitry than does a conventional cell. Even with the advent of very-large-scale integration technology, the single cell complexity still does not allow for the use of large associative memories.

(2) The memory storage provides poor storage density compared with conventional memory.

(3) The third major difficulty is the complexity of the interconnects. Recall that in order for all cells to compare their values to that of the comparand register, the control unit must broadcast the value to all cells involved in the comparison. However, using conventional technology, we see that the time delays associated with the broadcasting function are appreciable. Moreover, intercell interconnects become cumbersome for a large array size.

(4) The fourth difficulty is the lack of an efficient means of implementing parallel access to the cells, namely, parallel input and output.

III. Optical Content-Addressable Parallel Processor

Optical systems hold the promise of providing efficient support for future parallel processing systems. Optics advantages have been cited on numerous occasions.[11–18] These include inherent parallelism, high spatial and temporal bandwidths, and noninterfering communications. For associative processing, optics may be the ideal solution for the fundamental problems faced in electronic implementations; namely, cell complexity, interconnect latency, difficulty of implementing information broadcasting, and parallel access to the stored data.

A. Roles of Optics in Content-Addressable Memory Design

Optics can provide direct architectural support for CAM's in the following ways:

(1) Optics can alleviate the cell complexity by migrating the implementation of wiring and logic into free space.

(2) The high degree of connectivity available in free-space space-invariant optical systems, and the ease with which optical signals can be expanded (which allows for signal broadcasting) and combined (which allows for signal funneling) can also be exploited to solve the interconnect design and alleviate network latency problems.

(3) Optical and electro-optical systems can offer considerably more storage capacity (especially secondary storage) than pure electronic systems.[19–21]

(4) Finally, the multidimensional nature of optical systems can alleviate the input–output problem by providing parallel access to the stored–processed data.

Most current research in optical CAM's is related to neural network models.[22,23] Neural network models used as CAM's can produce correct results even when their inputs are only partially presented. Therefore they can be used to select the best matches when partial inputs are provided. However, these models require not only a radical departure from current architectures, but also a major change in the programmability of computers. This paper concentrates on the use of optical CAM's for traditional search (exact-match) and information retrieval applications that are encountered in database–knowledge-base processing, expert systems, and list and string processing. Our goal is to develop adequate architectural support for these applications by exploiting the unique advantages of optics without radical changes to current programming practices. The rationale is that the resulting architecture should be similar to current computer systems but expanded in capability, for there is a significant base of applications that can take advantage of such an expanded system.

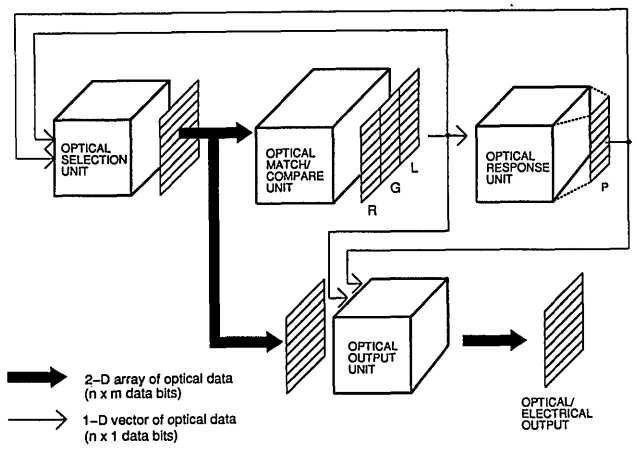

Fig. 2. Schematic organization of the optical content-addressable parallel processor.

B. Optical Content-Addressable Parallel Processor: Overview

In Fig. 2 a preliminary organizational structure for an OCAPP is proposed. The architecture is organized in a modular fashion and consists of a selection unit, a match–compare unit, a response unit, an output unit, and a control unit. The architecture is developed to meet four goals, namely, (1) exploitation of maximum parallelism, (2) amenability to optical implementation with existing devices, (3) modular design that can be scalable to bigger problems (to be explained later), and (4) capability of efficiently implementing information retrieval and symbolic computations. Moreover, the programming methodology for an OCAPP is compatible with that of existing single-instruction multiple data systems.[24] In what follows, we describe the role of each unit. Detailed optical implementation of an OCAPP is presented in Section V.

The selection unit is schematically described in Fig. 3. It consists of (1) a storage array of n words, each m bits long (in actuality, the storage array capacity is $n \times 2m$, since each bit position consists of a true bit w_{ij} and its complement $\overline{w_{ij}}$); and (2) word and bit-slice enable logic to enable–disable the words and–or the bit-slices that participate in the match operation and reset the rest. It is assumed that the storage array can be loaded in parallel and (if need be) read in parallel. Optical means for achieving this are discussed below.

The match–compare unit shown in Fig. 4 contains (1) a $1 \times m$ interrogation register I; (2) logic hardware to perform parallel bitwise comparison between the bits of the interrogation register and the enabled bits of the storage array; (3) two $n \times 1$ working registers, G and L, which are used for magnitude comparisons; (4) an $n \times 1$ response register R for displaying the result of the comparison; and (5) a single indicator bit called the match detector MD, which indicates whether

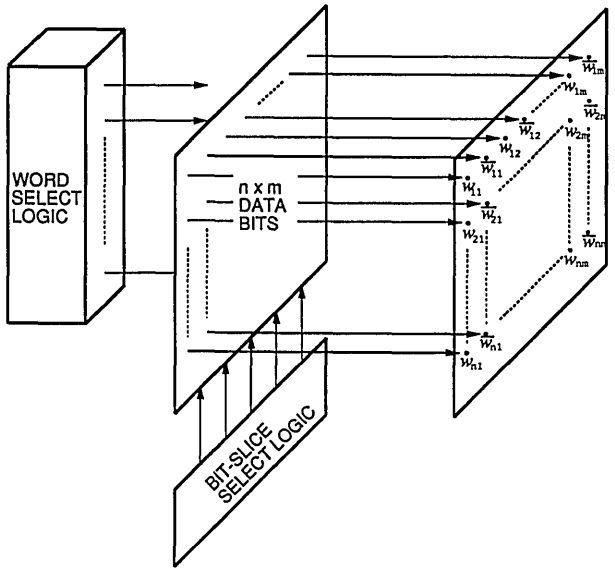

Fig. 3. Organization of the selection unit.

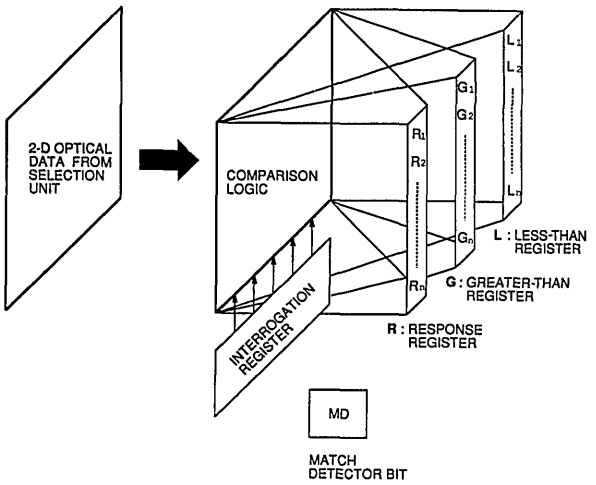

Fig. 4. Organization of the match–compare unit.

or not there are any matching words. This unit allows the comparison of a single operand stored in the interrogation register with all the words stored in the storage array. As such, it is considered an SIMD unit. Bit position R_i of R is set to one when word W_i of the storage array matches the contents of I. The I register is a combination of the comparand register C and the mask register M, as shown in Table 1. As such, it holds the operand (depending on masking information, if any) that is being searched for or is being compared with. It is assumed that register I is available in dual-rail logic (both true and complement bits are available).

The response unit is responsible for selecting the first matching word when there are several matching words. It consists of a combinatorial priority circuit for selecting the first matching word. Depending on the program control, the output of the response unit is either routed to the output unit for outputting the result or fed back to the selection unit for further processing of the matching words. All units are under the supervision of a conventional control unit with conventional storage (e.g., a local random-access memory), which stores the program instruction. Its role is to load and unload the storage array, set and reset various registers such as the I, R, G, and L registers of the match–compare unit, enable and disable memory words, perform conditional instructions, monitor the MD bit, and test program termination. In Section IV

Table I. Formulation of Interrogation Register I

Search Bit, c_j	Mask Bit, m_j	Interrogation Bits, $I_j\ \overline{I}_j$
0	0	01
1	0	10
0	1	00
1	1	00[a]

[a]The entry $I_j\ \overline{I}_j = 00$ means that bit position j for all words does not participate in the match operation (e.g., bit slice j is disabled).

we describe the implementation of several parallel algorithms on an OCAPP in order to show its implementation requirements and processing benefits. We then proceed to describe the optical implementation of its functional units in Section V.

IV. Parallel Search Algorithms on an Optical Content-Addressable Parallel Processor

We classify search operations as basic and compound operations. A basic search operation is one that can be completed in one sweep over all the bit slices of the storage array. It does not involve any feedback processing. A compound search operation requires a feedback from the response unit to the selection unit. As a consequence, it takes more than one sweep over the storage array to complete. For basic search operations, we group the following operations:

- The equivalence search comprises the equality search, the not-equal-to search, and the similarity search (search for a match within a masked field).
- The threshold search comprises the smaller-than, the not-smaller-than, the greater-than, and the not-greater-than searches.
- The extremum search comprises the greatest-value search and the smallest-value search.

Compound search operations can be implemented in a series of basic search operations. For the compound search, we group the following operations:

- The adjacency search comprises the next-above search and the next-below search.
- The between-limits search is the search for words z between two limits X and Y $(X < Y)$ for the conditions $X \le z \le Y, X < z \le Y, X \le z < Y$, and $X < z < Y$.
- The outside-limits search is the search for words z outside two limits X and Y $(X < Y)$ for the conditions $X \ge z \ge Y, X > z \ge Y, X \ge z > Y$, and $X > z > Y$.
- The ordered retrievals (sorting) are the ascending order retrieval and the descending order retrieval.

Of course, many more compound search operations can be formulated by using the basic search operations. The above search operations are the most frequently used in information retrieval applications.

A. Parallel Algorithms for Basic Search Operations on an Optical Content-Addressable Parallel Processor

In what follows, we denote a memory word as $W_i = (w_{i1} w_{im-1} \cdots w_{im})$, where w_{ij} is the jth bit cell of the word W_i, w_{i1} is the most significant bit, and w_{im} is the least significant bit. We denote the jth bit slice by $B_j = (w_{1j} w_{2j} \cdots w_{nj})$, which is made up of the jth bit of every word in the storage array. The interrogation and response registers are denoted by $I = I_1 I_2 \cdots I_m$ and $R = R_1 R_2 \cdots R_n$, respectively. The comparand word (search argument) and the mask register word are denoted by $C = (c_1 c_2 \cdots c_m)$ and $A = (A_1 A_2 \cdots A_m)$, respectively.

1. *Equivalence Search*

In this type of search, the memory is partitioned according to the magnitude of the search word C into two sets, namely, words that are equal to C and words that are not. The equality and masked search operations can be implemented by a bitwise match.[3] For an equality match, all the bits of the search word need to be matched, whereas for the masked search, only a subset of the bits of the search word is compared with the respective bits of the memory words. $A_j = 0$ means that c_j is not masked, while $A_j = 1$ means that c_j is masked. These two search modes can be combined as shown in Table 1. The comparison operation can be accomplished by using either the exclusive-OR (for bit mismatch) or the exclusive-AND (for bit match) logic functions.[3] In the proposed system, we consider the exclusive-OR function since it can be efficiently and easily implemented in optics.

Given a search word C (including masking information), we see that the exclusive-OR function for a bit mismatch denoted by $\overline{b_{ij}}$ on the jth cell of the ith memory word is

$$\overline{b_{ij}} = (I_j \wedge \overline{w_{ij}}) \vee (\overline{I_j} \wedge w_{ij}), \tag{1}$$

where the symbols $\wedge$, $\vee$, and the bar ($\overline{\ }$) denote the logical AND, the logical OR, and the logical NOT, respectively. Now the inequality of memory word W_i with interrogation vector I requires at least one bit mismatch between the two words, and therefore

$$M_i = \bigvee_{j=1}^{j=m} \overline{b_{ij}} = \overline{b_{im}} \vee \overline{b_{im-1}} \vee \cdots \vee \overline{b_{i1}}, \tag{2}$$

where the big $\vee$ denotes a logical OR over all bits and M_i indicates a word mismatch. Alternatively, the equality of words W_i and I is also computed by

$$R_i = \overline{M_i} = \overline{(I_1 \wedge \overline{w_{i1}}) \vee (\overline{I_1} \wedge w_{i1}) \cdots (I_m \wedge \overline{w_{im}}) \vee (\overline{I_m} \wedge w_{im})}. \tag{3}$$

Equation (3) indicates that matching words in memory will be flagged by having their corresponding R bit set to one, and all mismatches will have their R bits set to zero. Indicator bit R_i can be computed by using NOR logic on all the mismatch bits $\overline{b_{ij}}$ for $j = 1, \ldots, m$. Moreover, Eq. (3) is space invariant and can be implemented in bit-parallel and word-parallel fashion because of the inherent parallelism of optics. Consequently, all R_i's for $i = 1, \ldots, n$ can be computed at the same time with a single access to the storage array. Implementation details are described in Section V.

Equivalence Search Algorithm

(1) Initialize by using the following steps:

(a) Load I (this will depend on the search word and the masking condition).

(b) Clear R (clear all bits in the R register).

(2) Compute the match condition with $R_i = \vee_{j=m}^{j=1} \overline{b_{ij}}$ for $i = 1, \ldots, n$. ($R_i = 1$ if and only if W_i matches I.)

2. *Threshold Search*

This mode of search partitions the memory according to the magnitude of the interrogation vector I into three sets, namely, words equal to, less than, and greater than I. We introduce here an algorithm that provides this result by simultaneously using three registers of the response unit, namely, R, G, and L. We should note that there are other parallel algorithms that can accomplish the same effect.[6] Initially, all memory words are made active by setting registers RGL = 100 (for all bits). The memory is scanned from the most significant to the least significant bit position by enabling a single bit slice at a time. When the bit I_j is one, we select all active memory words with $w_{ij} = 0$ as "less than" by setting their corresponding bit position RGL = 001. These words are then disabled from further comparisons. Similarly, when $I_j = 0$, we select all active memory words with $w_{ij} = 1$ as "greater than" by setting their corresponding bit position RGL = 010, and then we disable them from further processing. At the end of the last bit position, words still in the state RGL = 100 are equal to the comparand, words in the state RGL = 010 are greater than the comparand, and words in the state RGL = 001 are less than the comparand. It is important to note that even though we are scanning the memory from the most significant bit to the least significant bit, the search process can be terminated any time there are no matching words at a given bit position ($R_i = 0$ for all $i = 1, \ldots, n$). Such a condition is easily detectable by checking the MD bit.

Threshold Search Algorithm

(1) Initialize by using the following steps:

(a) Load I (depending on the search word and masking condition).

(b) Enable memory words.

(c) Set R, clear G, clear L, and set $j = 1$ (the variable j is used by the control unit to scan the storage array).

(2) Perform the magnitude search at bit-slice j (compute R_i, G_i, and L_i). $R_i = \vee_{j=m}^{j=1} \overline{b_{ij}}$, $G_i = \overline{I_j} \wedge w_{ij}$, $L_i = I_j \wedge \overline{w_{ij}}$ for $i = 1, \ldots, n$ (note that only the enabled bit slice j determines the values of R_i, G_i, and L_i; all other bit slices are disabled at this time, and therefore have no influence on R_i, G_i, and L_i).

(3) Test whether MD = 1 (are there any words that match the I register at the current bit position j?), and then use the following procedures:

(a) If MD = 1, proceed to step 4.

(b) If MD = 0, proceed to step 6.

(4) Disable memory words whose corresponding bits in R are zero. Memory words whose corresponding bit $R_i = 0$ have already been decided on. Words with $L_i = 1$ are less than I, and words with $G_i = 1$ are greater than I. These words can be outputted at this time if needed.

(5) Increment j with $j \leftarrow j + 1$, test if $j = m$, and use the following procedures:

(a) If $j \neq m$, proceed to step 2.

(b) If $j = m$, proceed to step 6.

(6) The search is done and the result is reported in R, G, and L.

Table II illustrates the threshold search algorithm for a magnitude search of 7 words, each 5 bits long.

3. *Extremum Search*

This type of search refers to finding the maximum (or minimum) of a set of (or all) memory words. We consider first the search for the maximum.

a. Maximum Search. To find the maximum we can scan the memory words from the most to the least significant bit positions. As we scan the bit slices, we determine if any of the enabled words have a one in the current bit position. If we find some, we disable all those words that do not have a one in this position. If none of the words at the current position possesses a one, we keep scanning. At any given time, all remaining candidate words are equal as far as we have examined them, because for every bit position either every word has a zero in that bit position, or when some words have ones, we disable the ones with zeros. Therefore at bit position j, enabled words with $w_{ij} = 1$ are larger than enabled words with $w_{ij} = 0$. Since we are seeking the maximum, we disable the ones with $w_{ij} = 0$. This process is repeated until we exhaust all bit positions, at which time the maximum word will be indicated by $R_i = 1$.

Finding the Maximum Algorithm

(1) Initialize by using the following steps:

(a) Load I with $I \leftarrow 11 \cdots 11$ (I is loaded with all bits set to one).

(b) Set R, clear MD, and set $j = 1$.

(c) Enable memory words.

Table II. Example of Threshold Search Algorithm[a]

Memory Word, i		State of RGL at the End of Each Iteration					
		$j =$					
1	W_i	1	2	3	4	5	(Last Iteration)
1	10111	100	100	100	100	010	($W_1 > S$)
2	11000	100	010	010	010	010	($W_2 > S$)
3	10010	100	100	001	001	001	($W_3 < S$)
4	10110	100	100	100	100	100	($W_4 = S$)
5	10101	100	100	100	001	001	($W_5 < S$)
6	01101	001	001	001	001	001	($W_6 < S$)
7	11101	100	010	010	010	010	($W_7 > S$)

[a]Search word S, 10110; mask word, 00000; I register, 10110 (effective word search).

(2) Perform an equivalence search at bit slice j (compute R_i for all $i = 1, \ldots, n$).

(3) Test whether MD = 1 (are there any words with a one in the current bit position j?), then use the following procedures:

(a) If MD = 1, proceed to step 4.

(b) If MD = 0, proceed to step 6.

(4) Disable all words that do not have a one in the current bit position (these words are indicated by $R_i = 0$).

(5) Clear R and MD.

(6) Increment j with $j \leftarrow j + 1$, test if $j = m$, and choose one of the following:

(a) If $j \neq m$, go to step 2.

(b) If $j = m$, the output maximum value is indicated by $R_i = 1$.

Table III illustrates the maximum search algorithm.

b. Minimum Search. The search for the minimum is very similar to the search for the maximum except that the I register is initially loaded with zeros, and if any enabled word has a zero in the current bit position (there exists a memory word W_i such that its corresponding $R_i = 1$), we disable the words with a one in the current bit position ($R_i = 0$). These words have to be greater than the minimum sought. The process is repeated until we exhaust all bits of the enabled words. The minimum value will also be indicated by a one in register R.

B. Parallel Algorithms for Compound Search Operations on the Optical Content-Addressable Parallel Processor

Compound search operations such as the ones stated in Subsection IV.A. cannot be economically implemented by a single sweep over the memory words. We therefore choose to implement such operations as a series of basic searches. The rationale is to keep the architecture as simple as possible, and therefore make it highly amenable to optical implementation. Of course, speed improvements can be gained by implementing these search operations as a basic search, but the number of logic circuits may be extensive.

Table III. Example of Maximum Search Algorithm

	State of Register R at the End of Each Iteration					
	j =					
Memory Word, i	1	2	3	4	5	(Last Iteration)
11000	1	1	0	0	0	
11100	1	1	1	0	0	
10001	1	0	0	0	0	
11110	1	1	1	1	1	(Maximum)
11001	1	1	0	0	0	

1. Double-Limit Search (Between and Outside Limits)

Given two numbers called HIGH and LOW, we see that the double-limit search consists of finding those words that are between these limits and–or words that are outside these limits. This gives rise to eight different searches that can be accomplished in a very similar manner. Let us consider the between-limit search. Given the two numbers HIGH and LOW, we wish to find those words that are greater than LOW but less than HIGH, namely, all W_i such that LOW $< W_i <$ HIGH. We can accomplish this search by using the threshold search described previously, as follows. First, we determine the words that are less than the comparand HIGH. These words will be indicated by a one in the L register. We then disable all other words except the ones that are less than HIGH, and perform another threshold search by using the comparand LOW. After the second search, words that are less than HIGH and greater than LOW will be marked with a one in the G register, which could be routed to the output unit for outputting the search result. The algorithm follows.

Between-Limits Search Algorithm.

(1) Initialize by using the following steps:

(a) Load I with comparand HIGH.

(b) Enable all memory words (initially all memory words participate in the search).

(c) Set R, clear G, clear L, and set $j = 1$.

(2) Perform the threshold search.

(3) Disable memory words with $L_i = 0$. Words with $L_i = 0$ are either greater than or equal to comparand HIGH and therefore need to be disabled at this time.

(4) Load I with comparand LOW.

(5) Perform the threshold search (at the end of this step, all words that are greater than LOW and less than HIGH will be marked with a one in the G register).

(6) Route the G register to the output unit (bit $G_i = 1$ of register G indicates that the memory word W_i satisfies LOW $< W_i <$ HIGH).

2. Adjacency Search

To find the word that is next above the comparand (the smallest word larger than the comparand), we search for all words that are larger than the comparand and then select the minimum. Similarly, to find the word that is next below the comparand (the largest word smaller than the comparand), we search for all words less than the comparand and select the maximum.

Next-Above Search Algorithm

(1) Initialize by using the following steps:

(a) Load I with a search word.

(b) Enable memory words.

(c) Set R, clear G, clear L, and set $j = 1$.

(2) Perform the threshold search (get all words greater than the comparand).

(3) Disable those words that are less than the

comparand (these words will be indicated by a zero in the G register).

(4) Perform a minimum search on all enabled words. The word next above the comparand is indicated by a one in the R register.

The search for the largest word smaller than the comparand (next-below search) can be carried out by an algorithm similar to the one above. In this case, step 2 of the next-above algorithm is replaced by a search for words that are less than the comparand, and step 4 is replaced by a maximum search.

3. Ordered Retrieval (Sorting)

Sorting a set of data using an OCAPP is straightforward. This can be achieved by performing the extremum search repeatedly until all the data are retrieved. For the ascending order retrieval we enable the memory words to be sorted and determine the minimum (by using the minimum search operation). We output the obtained minimum value and disable it from the storage array. We repeat these steps until we retrieve (in ascending order) all the enabled words. For descending order retrieval we select the maximum value at each step.

Ascending–Descending Order Retrieval Algorithm.

(1) Enable the set of words to be ordered.

(2) Perform a minimum–maximum search (determine the smallest or the largest word for descending order retrieval).

(3) Route R to the output unit (output the minimum–maximum word).

(4) Disable the selected minimum–maximum word.

(5) Repeat steps 2–4 until all enabled words are exhausted.

V. Optical Implementation

In Section V we identify the fundamental and basic operations required to implement the optical architecture and describe possible optical components for achieving them.

A. Basic Operations and Hardware Components Required

An analysis of the conceptual OCAPP and of the algorithms that have been described in Sections III and IV, respectively, reveals that the following is required:

(1) Data bits are in optical form and must be available in dual-rail format (both the value and its complement are required).

(2) Parallel access is required for writing into and reading from the storage array and the various control registers such as the interrogation register and the response register. This also includes selective writing into and selective reading from the memory of a single word or several words.

(3) Disablement of a memory word (or several memory words) whose corresponding bit in the response register (or the G register or the L register) is not asserted.

(4) Disablement of a memory word whose corresponding bit in the priority register P is asserted.

(5) Test the match detector bit MD.

(6) Optical array logic functions.

(7) Space-invariant optical interconnects.

(8) Optical broadcasting and funneling.

(9) Optical feedback connections.

(10) Dynamic routing of information (e.g., routing contents of register R to a selection unit, an output unit, or a response unit, depending on the algorithm).

The optical components required to accomplish the above operations can be divided into logic elements, storage elements, and information transfer elements (or interconnects).

For optical logic and storage, many approaches are being investigated. One approach is the adaptation of the spatial light modulator (SLM) technology to optical logic.[25–27] Another approach for realizing optical components capable of performing logic is to optimize the device from the beginning for digital operations. The recent emergence of the quantum-well self-electro-optic effect device (SEED) and its derivative, the symmetric or S-SEED, are such products.[28,29] The SEED devices can be used to realize logic operations such as NOR, OR, AND, and NAND, and they can be used for storage such as set–reset (S–R) latches.[30] Optical resonators are another family under this approach that are intended for optical logic.[31,32] Recently, a promising new device called the surface-emitting laser logic (CELL) device has been introduced both as an optical logic and as a latching device.[33] These devices are claimed to be ideal for applications that require high-optical gain, cascadability, and insensitivity to external optical feedback.

All data movements and information transfer in an OCAPP are space invariant, which may render their implementation easier. Classical optical components such as lenses, mirrors, beam splitters, holographic deflectors, and delay elements are most likely to be used for this purpose.[34] In addition, half-wave plates, shutters, and masks may be used for dynamic routing.[35–38]

B. Modular Implementation of an Optical Content-Addressable Parallel Processor

In this paper we present a modest design example of an OCAPP, and we use existing optical hardware in order to highlight the potential implementation issues of a practicable realization. The implementation of this first version will make use both of the SEED device for optical logic operating as a NOR gate, and of the S-SEED device operating as a S–R latch for storage.[30] The NOR gate is preferable to any other form of thresholding because it requires only discernment between the state in which no light comes in and the state in which light comes in. Thus a NOR gate

requires a signal-to-noise ratio better than one only. The family of SEED devices seems to be easy to use, and the devices are capable of high speed, low-energy operation, and realization in a two-dimensional format.[39] Space-invariant optical interconnects, dynamic masking components, and beam spreading and combining components are assumed for data routing.

A schematic diagram of the S-SEED device operating as an S–R latch is shown in Fig. 5(a). The state of the device is set by a pair of unequal signal beams labeled S (used for setting the output $Q = 1$, $\overline{Q} = 0$) and R (used for resetting the output $Q = 0$, $\overline{Q} = 1$). The device is set ($Q = 1$) when the power incident on the S input is much higher than the power incident on the R input. The state of the device is read by applying two equal-power (clock-signal) beams to both inputs. During the setting of the device, the clock beams must be low in comparison to the signal beams. The device holds its state when no clock signal is incident. Thus the device can operate as a latch. Moreover, during the application of the clock signal (the reading process), the state of the device is unaltered as shown in Fig. 5(b).

As described earlier, the optical processor can be constructed from several units: the selection unit, the match–compare unit, the response unit, the output unit, and the control unit. In Subsections V.C–V.G we describe the optical implementation (architectural rather than experimental setup) of each of these units. Moreover, the details in the routing and imaging paths, such as lenses, holographic elements, masks, beam splitters, and polarizers, as well as power supplies, presetting, and beam generation for the SEED, have been omitted from the diagrams to assist the reader's conceptual understanding of these configurations. The details of the actual use of SEED devices as optical logic elements can be found in Refs. 39–41. For diagrammatic simplifications, the SEED and S-SEED devices used in the following figures are assumed to be transparent rather then reflective.

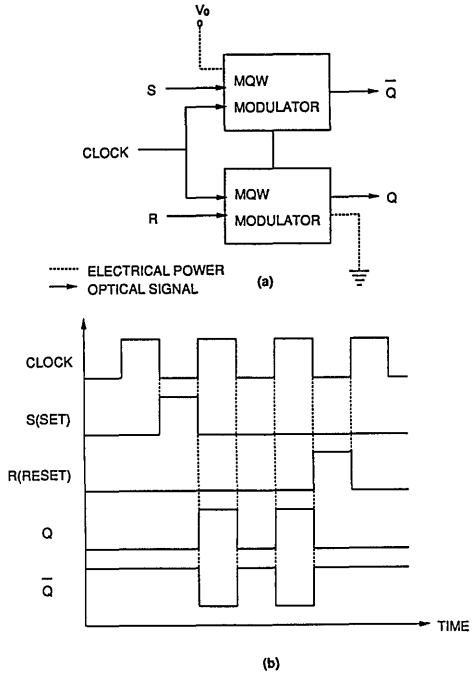

Fig. 5. S-SEED device operating as a clocked S-R latch: (a) schematic diagram, (b) timing diagram. MQW, multiple quantum well.

C. Optical Selection Unit

The optical selection unit of Fig. 6 is composed of a storage array, which consists of a two-dimensional $n \times (m + 1)$-bit array of clocked S-SEED devices (each entry in the array at position i, j has two incoming bits S, R and two outgoing bits w_{ij}, $\overline{w_{ij}}$); a clocked $n \times 1$-word register A, which serves to write data words into the storage array; and a clocked $1 \times m$-word register B for writing a single bit slice into the storage array. The first column of the storage array is reserved for an $n \times 1$-bit enable register ER, whose bit $ER_i = 1$ if and only if memory word W_i is enabled (more on enabling and disabling memory words will follow). The storage array can be directly loaded with input data in two-dimensional optical form from an external optical memory storage such as a page-oriented holographic memory,[20] or from a regular electronic memory if electrically addressed SLM's (E-SLM's) are used at the input side of the processor.

1. *Writing a Word–Bit Slice Into the Storage Array*

The storage array is assumed to be loaded in parallel at the beginning of the program. During program execution, the contents of the storage array can be altered by the use of the A and the B registers. The latter can be implemented by using E-SLM's such as the magneto-optic modulators (SIGHTMOD[42]) or the CCD-based liquid-crystal valves.

To write a word in the storage array, say at word position i, we first write the word in E-SLM B. In the next clock cycle, the contents of the B register are spread out vertically by using vertical spreading optics such that each bit B_j impinges on the set ports of the jth column of the storage array. Next, bit A_i of E-SLM A (corresponding to word position i) is pulsed high and spread out horizontally such that it impinges on the set and reset ports of the ith row of the storage array. A one bit is written in bit position w_{ij} of the storage array if and only if a high A_i and a high B_j coincide at the set port of bit w_{ij}. Similarly, a zero bit is written in bit position w_{ij} if and only if a high A_i and a high $\overline{B_j}$ coincide at the reset port of w_{ij}. This of course assumes that the set–reset thresholds of the S-SEED devices are so designed. Similar operations take place for writing a bit slice in the jth column of the storage array, with the exception of interchanging the roles of the B and A registers.

2. *Enabling–Disabling Memory Words*

By enabling a memory word W_i is meant including it in the matching process. Similarly, by disabling it is meant the excluding of it from further matching

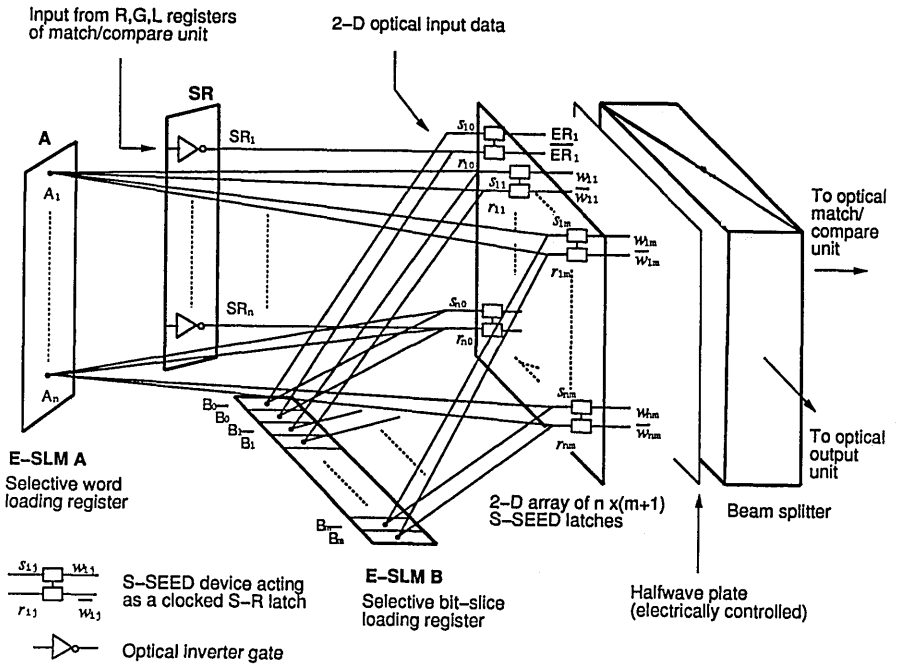

Fig. 6. Optical implementation of the selection unit.

operations. In order to have this capability, the first column of the storage array is designated as an enabling–disabling register ER. Bits of the ER register can be simultaneously set and reset, respectively, by using two external signals S-ER for setting all bits of ER, and by using R-ER for resetting all bits of ER (bits B_0 and $\overline{B_0}$ can be used for this purpose). Alternatively, selected bits of ER can be set and reset under program control by using an $n \times 1$ SR register (shown in the Fig. 6), which can be loaded from the R, G, or L registers of the match–compare unit. To enable (disable) the entire memory words, the S-ER (R-ER) bit is set and spread out vertically to all the set (reset) ports of the ER register. To selectively disable memory words whose R, G, or L bits are not asserted ($R = 0$, $G = 0$, or $L = 0$) requires the routing of the appropriate register (R, G, or L) to SR, which inverts its light intensity and routes it to fall upon the reset ports of ER. Thus word W_i of the storage array is disabled from further processing if light emanating from bit position $SR_i = 1$. For example, to disable memory words whose R bits are not asserted ($R_i = 0$) from further matching operations, we first route the contents of R to SR, which complements these data and images the complemented data upon the reset ports of SR. With a low bit $R_i = 0$, the ith output ER_i of the ER register is also set low, which in turn disables memory word W_i from participating in further comparisons. The use of the ER register in match operations is explained next.

The output of the selection unit can be routed to the match–compare unit for further processing, or to the output unit for outputting the result. Such routing can be achieved by using polarization beam splitters and electro-optical components that are capable of exchanging the state of polarization under electrical control, as, for example, the Pockel's cells.[34]

D. Optical Match–Compare Unit

This is the most critical unit in an OCAPP since it performs the match and magnitude comparison searches between the data stored in interrogation register I and words of the storage array. These operations should be implemented as parallel and as fast as possible. As shown in Fig. 7, the unit contains SEED arrays for comparison logic and three registers, namely, the response register R, the greater-than register G, and the less-than register L. Parallel comparison takes place between memory words emanating from the storage array and the interrogation register I, as explained below in Subsection V.D.1.

1. Optical Implementation of Parallel Match and Comparison Operations

A match at bit w_{ij} is detected by an exclusive-OR principle as indicated by Eq. (1), and a word match is computed according to Eq. (3), which is rewritten here by using only the NOR function, as

$$R_i = \overline{\overline{I_1 \vee \overline{w_{i1}}} \vee \overline{\overline{I_1} \vee w_{i1}} \cdots \overline{I_m \vee \overline{w_{im}}} \vee \overline{\overline{I_m} \vee w_{im}}}, \qquad (4)$$

and the relative magnitude comparison between the interrogation word and stored data is given by

$$G_i = \overline{I_j} \wedge w_{ij} \qquad (\text{word } W_i > I), \qquad (5)$$

$$L_i = I_j \wedge \overline{w_{ij}} \qquad (\text{word } W_i < I). \qquad (6)$$

Note that the magnitude comparison is performed bit serially; therefore bits L_i and G_i require a single bit

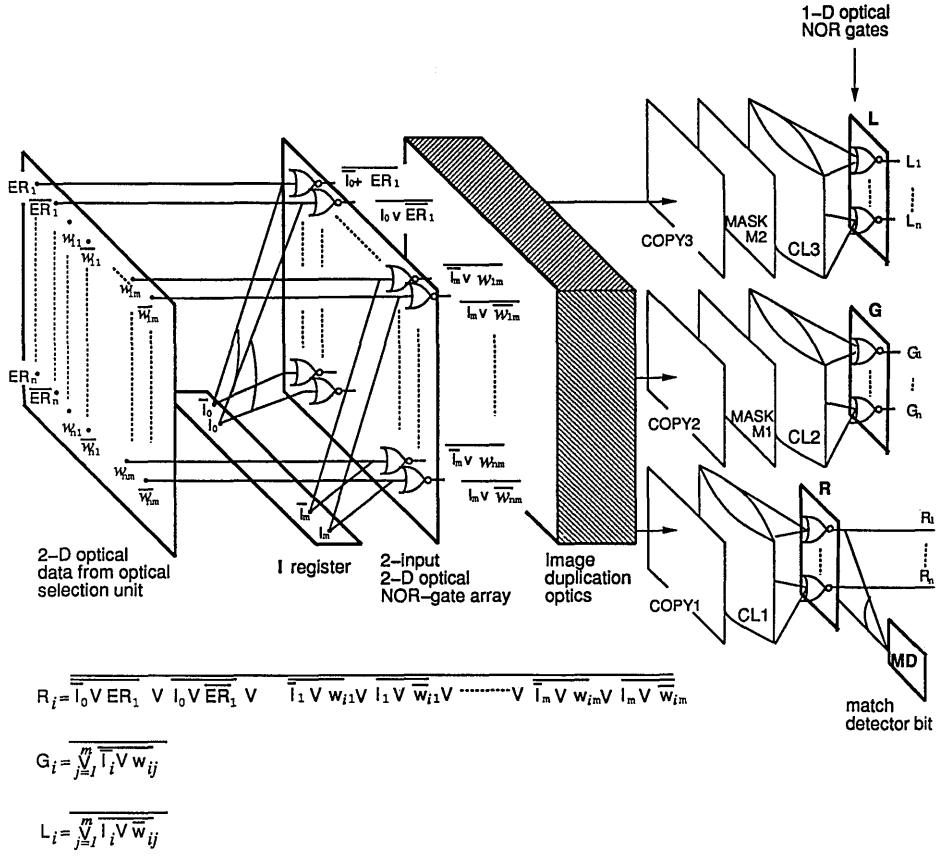

Fig. 7. Optical implementation of match–compare unit.

comparison. In the best case (when there are no matching words) it takes a single bit comparison (the most-significant bit position) to determine the values of L_i and G_i. In the worst case, Eqs. (5) and (6) are computed m times.

In order to have the capability of selectively disabling memory words from further matching, we need to incorporate the contents of the enabling register ER in the match condition described above. One way of including ER in Eq. (4) would be

$$R_i = \overline{\overline{M_i} \vee \overline{ER_i}} \qquad \text{for all } i = 1, \ldots, n. \tag{7}$$

Whenever ER_i is zero, R_i is zero, and therefore word W_i is disabled. The optical implementation of Eq. (7) would require that each bit ER_i of register ER be split into two light beams and that both be directed to a single NOR gate of the two-input NOR-gate array of the match–compare unit. While this is feasible, it would require space-variant interconnections between the optical selection unit and the optical match–compare unit since the rest of the data (e.g., memory words) in the storage array are simply imaged upon the two-dimensional NOR-gate array.

In order to keep the interconnections space invariant and to be able to selectively disable memory words from matching the interrogation register, we formulate the match condition as follows:

$$R_i = \overline{\overline{M_i} \vee (I_0 \wedge \overline{ER_i} \vee \overline{I_0} \wedge ER_i)} \qquad \text{for all } i = 1, \ldots, n. \tag{8}$$

The optical implementation of the above equation does not require any special interconnection patterns between the selection unit and the match–compare unit because the ER register is treated as any other column of the storage array. Therefore any imaging system would route the data, including ER, to the match–compare unit. However, the interrogation register needs to be $(m + 1)$-bits long. The extra bit I_0 is set to one during a match–compare operation. Eq. (8) can also be expressed in terms of the NOR function only as follows:

$$R_i = \overline{\overline{I_0 \vee \overline{ER_i}} \vee \overline{\overline{I_0} \vee ER_i} \vee \overline{\overline{I_1 \vee \overline{w_{i1}}} \vee \overline{\overline{I_1} \vee w_{i1}} \cdots \overline{\overline{I_m} \vee \overline{w_{im}}} \vee \overline{\overline{I_m} \vee w_{im}}}}. \tag{9}$$

It can easily be verified that when $ER_i = 0$, the above equation yields $R_i = 0$, and when $ER_i = 1$ bit, R_i depends on the magnitudes of the search word I and

the memory word W_i. Optical hardware to implement the above equation is shown in Fig. 7. In order to implement parallel comparison, the bits of register I need to be spread out vertically so that each bit I_j ($\overline{I_j}$) impinges on one port of the NOR gates of the jth column of the two-dimensional NOR-gate array, while data bits $\overline{w_{ij}}$ (w_{ij}) for $i = 1, \ldots, n$ impinge on the second ports of the same NOR gates of the two-dimensional NOR-gate array. The output of the two-dimensional optical NOR-gate array is replicated into three copies. Cylindrical lens CL_1 positioned in the path of the first copy collects the output of an entire row i into a single position i of a second one-dimensional NOR-gate array that represents register R. The second copy is passed through a fixed mask M_1 whose purpose is to block part of the information that is contained in the replicated copy, namely, the term $\overline{I_j \vee w_{ij}}$, and let pass the term $\overline{\overline{I_j} \vee w_{ij}}$ for all $i = 1, \ldots, n$ and $j = 0, \ldots, m$. The unmasked output is collected by cylindrical lens CL_2 into a single one-dimensional optical NOR-array, which constitutes register G. Similarly, the third copy is passed through a fixed mask M_3, which blocks the terms $\overline{\overline{I_j} \vee w_{ij}}$ and lets through the terms $\overline{I_j \vee \overline{w_{ij}}}$. The unmasked data are collected into a single one-dimensional optical NOR-gate array by cylindrical lens CL_3 to form the L register.

A single level of light intensity on the input side of bit position R_i will indicate that the word W_i of the storage array and search word I differ in at least one bit. Inversely, if zero light impinges on position i of the one-dimensional NOR-gate array implementing the R register, this will indicate that the two words are equal. Similarly, a single level of light intensity on the input side of registers G and L will set the appropriate bit. Although it appears that bits L_i and G_i of registers L and G in Fig. 7 depend on all the bits in row i (after appropriate masking) of the output of the two-dimensional NOR-gate array, in actuality that is not the case. Recall that registers L and G are used only in bit-serial algorithms in which a single bit slice of the storage array is enabled at a time. Therefore only a single bit in row i of the storage array determines the values of bits L_i or G_i (see Section III for details).

Finally, all the bits of register R are logically OR'ed into a single photodetector cell to form the match detector MD bit. The MD flip-flop is a quick indication of whether or not there are any matches between the contents of register I and memory words. The output of the match–compare unit (e.g., the contents of the R, G, and L registers) can be routed to the response unit for selecting the first matching word, or to the output unit for outputting the relevant data, or fed back to the selection unit for disabling irrelevant words in bit-serial processing. Such routing is accomplished by beam splitters, imaging optics, space-invariant optical interconnects, and control signals emanating from the control unit.

2. *Two-Dimensional Optical Matching*

The optical match–compare unit of Fig. 7 consists of a single interrogation register I, and therefore allows comparison of one search argument with the words stored in the storage array. However, because of the multidimensionality of optical systems, this unit can be extended to perform multiple search operations in a single step. That is, several search arguments are compared simultaneously with the words of the storage array. An extended multiple-interrogation match detector match–compare unit would have a $k \times m$ two-dimensional array of k search arguments, a two-dimensional storage array of n words each m bits long, and an $m \times k$ two-dimensional response array as shown in Fig. 8. Each response register R_l ($l = 1, \ldots, k$) would indicate the match between

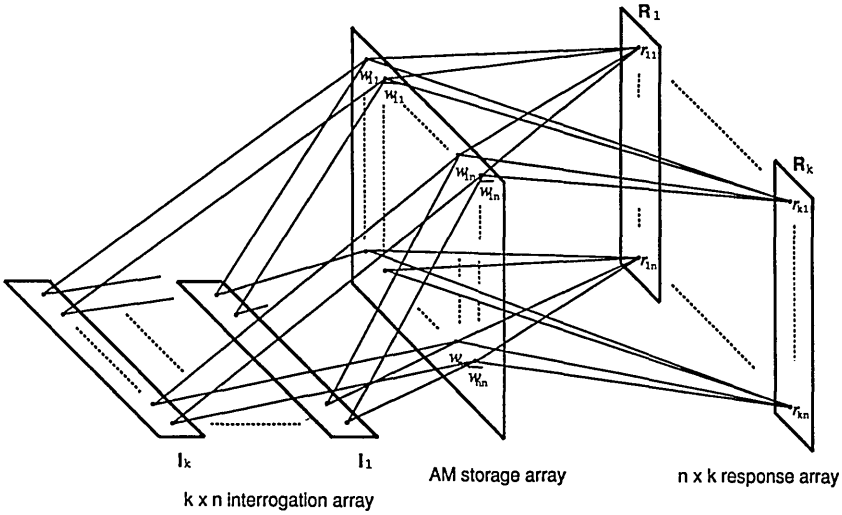

Fig. 8. Optical implementation of a multiple match–compare unit. The interrogation and the response registers of Fig. 7 are replaced by two-dimensional arrays of search arguments and response registers, respectively. Register R_1 indicates the match or mismatch of memory words with interrogation register I_1 (for i = 1, . . . , k).

interrogation register I_l ($l = 1, \ldots, k$) and the words of the storage array. The two-dimensional match operation can be thought of as an optical binary matrix–matrix multiplication that can be implemented by using several optical techniques.[43–45]

E. Optical Output Unit

The output unit outputs memory words whose corresponding bits in the R, G, L, or P registers are asserted. This unit should have the capability to output a single memory word or several words at a time. For a single word output, the output word is obtained by a multiplexing operation,

$$\overline{O_j} = \overline{P_1 \wedge w_{1j} \vee P_2 \wedge w_{2j} \vee \cdots \vee P_n \wedge w_{nj}} \qquad \text{for } j = 1, \ldots, m. \tag{10}$$

The above equation can be expressed in terms of the NOR function as follows:

$$\overline{O_j} = \overline{\overline{P_1} \vee \overline{w_{1j}} \vee \overline{P_2} \vee \overline{w_{2j}} \vee \cdots \vee \overline{P_n} \vee \overline{w_{nj}}}. \tag{11}$$

Similarly,

$$O_j = \overline{\overline{P_1} \vee w_{1j} \vee \overline{P_2} \vee w_{2j} \vee \cdots \vee \overline{P_n} \vee w_{nj}}. \tag{12}$$

Figure 9(a) depicts the optical implementation of Eqs. (11) and (12). The priority register is inverted with an $n \times 1$ SEED inverter array, denoted by N. Each output bit of register N is expanded in the horizontal direction and imaged upon one port of a two-input two-dimensional optical NOR-gate array. The contents of the storage array are imaged upon the second port of the two-dimensional NOR-gate array. The output of the two-dimensional NOR-gate array is vertically collected by cylindrical lens CL_1 to fall upon the one-dimensional optical NOR-gate array, which represents the desired selected output word. Parallel readout of several words can also be accomplished as shown in Fig. 9(b). The optical two-input AND-gate array reflects the superposition of a data page from the storage array and a data page formed by the horizontally expanded register T, which can be loaded from registers R, G, or L. The superposed image is directed by beam splitter BS_2 to a two-dimensional detector array for the parallel readout of desired words. It should be noted that Fig. 9(b) can also be used to output a single memory word if row-addressable two-dimensional photodetectors are used.

F. Optical Response Unit

The response unit contains a combinational priority circuit, and it contains a priority register P for indicating the first matching word in memory. The priority circuit allows only the first responder (the first memory word W_i whose R_i is one) to pass to the priority register P.

The priority circuit can be implemented by using several stages of one-dimensional NOR-gate arrays in the form of a binary tree with space-invariant interconnections between them.[46] The size of the NOR-gate arrays is equal to the number of words n stored in the storage array, while the number of stages is proportional to $\log_2(n)$. A good technique for implementing such a unit would be the logic-interconnect architecture introduced by Murdocca *et al.*[36] and Murdocca.[47] The contents of the resulting P register are routed to the output unit for outputting a single word and also are fed back to the selection unit for enabling and disabling purposes.

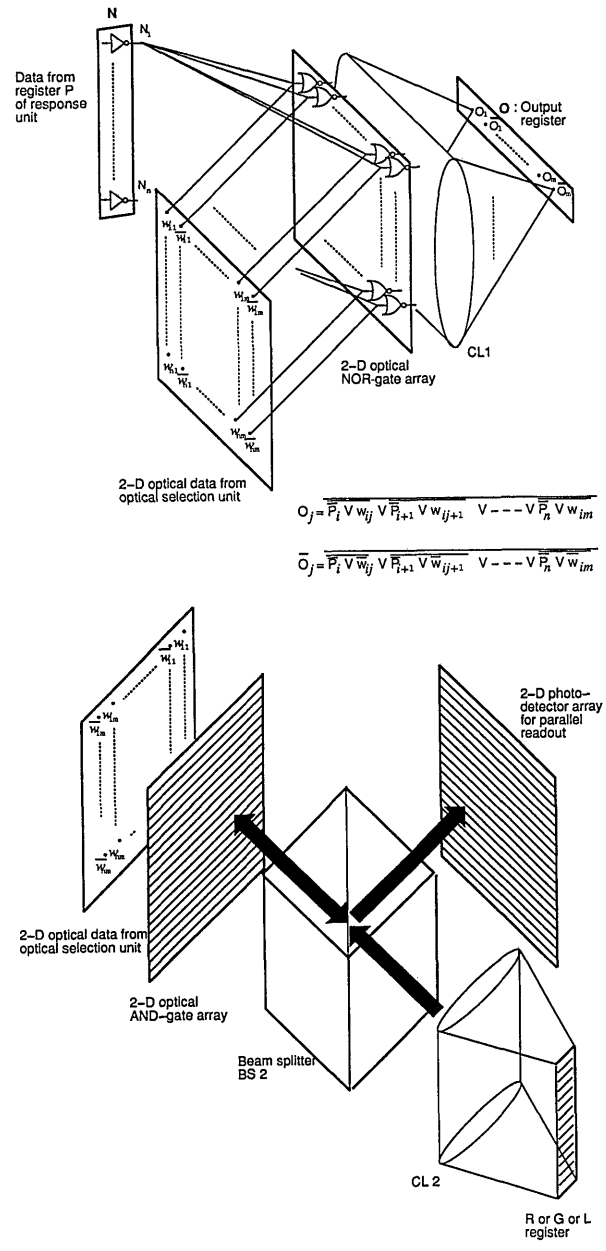

Fig. 9. Optical implementation of the output unit: (a) single word output, (b) parallel readout of multiple words.

G. Control Unit

The OCAPP is under the control of a memory control unit, which comprises a local memory for storing

programs and a program sequencer for executing instructions that control the optical hardware such as the S–R latches, the NOR-gate arrays, the routing shutters, and the splitters. The instruction set is composed of conventional assignment and conditional statements, and it is composed of additional instructions required to implement associative parallel processing. This includes data movement between units, comparison operations, memory loading and unloading, and monitoring of the MD bit. These additional instructions are few and are derived from the required fundamental operations described above. It should be noted that application programs for OCAPP can be written in conventional high-level languages such as Pascal or C, with few calls to external procedures that support parallel associative processing.

VI. Performance Analysis

An exact performance analysis of the proposed OCAPP, including speed, cost, and power-budget breakdown, is currently not feasible. We therefore try to estimate theoretically the execution time of the various algorithms presented. For the current analysis, we will not estimate the power required for the system. The following are definitions of terms and assumptions used in the time complexity analysis:

(1) The storage array consists of $n(m + 1)$-bit-long words that require an S-SEED array of $n \times 2(m + 1)$ pixels (recall that the first column is reserved for ER and that dual-rail coding is used).

(2) The response time of each algorithm is the sum of the following three terms: setup time T_{setup}, execution time T_{ex}, and transfer time T_{transf}. The setup time includes the time taken to load the data into the S-SEED array and the interrogation word into the E-SLM, the time taken to clear control registers, and the time taken to initialize the system's various components, such as shutters and electro-optical components for routing purposes. The execution time is the time taken to compute the desired result, and the transfer time is the time taken to transfer the computed result to the front-end or host computer that drives the OCAPP. In this paper we only estimate execution time.

(3) We assume that the response times of the optical S-R latches (S-SEED's) and that of the various logic arrays such as the NOR-gate arrays, the AND-gate arrays, and the optical inverters are comparable and are all equal to T_{resp}.

(4) T_p is light propagation time through a $4f$ imaging system.

(5) T_i denotes the time required to load the interrogation register I. Register I can be loaded in parallel in a single step since it is assumed to be a one-dimensional electrically addressed modulator.

(6) It is also assumed that reading memory words from the storage array and reading the I register is done at the same time and takes T_{resp}. The enabling and disabling of memory words are achieved in T_{resp} time, and testing the MD bit takes T_{resp}.

A. Optical Content-Addressable Parallel Processor Cycle Time and Estimated Execution Time for the Algorithms

1. *OCAPP Cycle Time*

The cycle time of a computing machine is defined to be the time required for the shortest well-defined processor micro-operation.[9] For the OCAPP, we define the cycle time T_{proc} to be the time required for the equivalence search operation (optical comparison) since the latter is at the core of all the algorithms intended for the OCAPP. This time includes (1) reading out stored data and propagating them to the match–compare unit, (2) using the NOR function on the stored data with the contents of the interrogation register, and (3) propagating the output of the NOR-gate array and producing the result in the R, G, and L registers, as shown by

$$T_{\text{proc}} = \overbrace{T_{\text{resp}} + T_p}^{1} + \overbrace{T_{\text{resp}}}^{2} + \overbrace{T_p + T_{\text{resp}}}^{3} = 3T_{\text{resp}} + 2T_p. \tag{13}$$

The numbers over the braces indicate the times needed to accomplish each subtask as enumerated above. The dominant factor in Eq. (13) is the response time of the SEED arrays used for storage and logic. Although these devices have been demonstrated to switch with speeds in the picosecond range (for a single device), the required optical switching power prohibits the use of larger arrays at high speeds. Currently, a single S-SEED gate requires approximately 2.5 pJ per switching event. Therefore if we were to use a 256 × 256 array of these gates (this array size has recently been reported), running at 100 MHz would require laser power in the kilowatt range. This is excessively more power than is available from most visible lasers. Moreover, these devices are sensitive to small temperature changes, which make them unreliable at these speeds. However, intensive research efforts are being pursued to lower the energy power requirements for larger array sizes of these devices and for solving the problems of focusability, operating wavelength, and stability at higher switching speeds.

A promising alternative to the SEED family of devices is the cascadable optical logic devices called surfaCe-Emitting Laser Logic (CELL) devices.[33] These devices are expected to have a much more tolerant operating range than SEED's. The CELL's are expected to have a broadband input and will operate in the gigahertz range. Of course, the proposed system will benefit from advances in other optically addressed SLM's as well.

2. *Execution Time for Threshold Search Algorithm*

The time it takes for the completion of one iteration of the threshold algorithm includes (1) reading out

stored data and propagating it to the match–compare unit, (2) using the NOR function on the stored data with the contents of the interrogation register, (3) propagating the output of the NOR-gate array and producing the result in the R, G, and L registers, (4) testing the MD bit, and (5) propagating the contents of R to the selection unit and appropriately setting the ER register. This process is either repeated as many times as the word length m or terminated any time the MD bit yields a zero light detection. The best-case execution time is then

$$T_{\text{tresh}} = \overbrace{T_{\text{resp}} + T_p}^{1} + \overbrace{T_{\text{resp}}}^{2} + \overbrace{T_p + T_{\text{resp}}}^{3} + \overbrace{T_{\text{resp}}}^{4}, \tag{14}$$

and the worst-case execution time is

$$T_{\text{tresh}} = m(\overbrace{T_{\text{resp}} + T_p}^{1} + \overbrace{T_{\text{resp}}}^{2} + \overbrace{T_p + T_{\text{resp}}}^{3} + \overbrace{T_{\text{resp}}}^{4} + \overbrace{T_p + 2T_{\text{resp}}}^{5}). \tag{15}$$

The numbers over the braces indicate the time needed to accomplish each subtask as enumerated above. Note that the loading time for register I after each iteration is omitted from the maximum execution time expression since this can be overlapped with some other activities.

3. *Execution Time for Extremum Search Algorithm*

The algorithm for the extremum search is quite similar to that of the threshold search algorithm. Therefore its execution time is equal to that of the worst-case execution time for the threshold search,

$$T_{\text{extrem}} = m(6T_{\text{resp}} + 3T_p). \tag{16}$$

4. *Execution Time for Double-Limit Search Algorithm*

As described earlier, the double-limit search algorithm is implemented by using the threshold search twice. The major processing steps are (1) searching for words less than the upper limit, (2) disabling the words that are greater than the upper limit (words with corresponding $L_i = 0$), (3) loading the lower limit in the I register, and (4) searching for words that are greater than the lower limit. Thus the best-case execution time is

$$T_{\text{double}} = \overbrace{4T_{\text{resp}} + 2T_p}^{1} + \overbrace{T_p + 2T_{\text{resp}}}^{2} + \overbrace{T_i}^{3} + \overbrace{4T_{\text{resp}} + 2T_p}^{4}, \tag{17}$$

and the worst-case execution time is

$$T_{\text{double}} = \overbrace{m(6T_{\text{resp}} + 3T_p)}^{1} + \overbrace{T_p + 2T_{\text{resp}}}^{2} + \overbrace{T_i}^{3} + \overbrace{m(6T_{\text{resp}} + 3T_p)}^{4}. \tag{18}$$

5. *Execution Time for Adjacency Search Algorithm*

The adjacency search comprises three major consecutive steps; (1) a threshold search, (2) a disabling operation, and (3) an extremum search (either a search for a minimum value in the case of the next-above search algorithm, or a search for a maximum value for the next-below search). Therefore its best-case execution time is

$$T_{\text{adjac}} = \overbrace{4T_{\text{resp}} + 2T_p}^{1} + \overbrace{T_p + 2T_{\text{resp}}}^{2} + \overbrace{m(6T_{\text{resp}} + 3T_p)}^{3}, \tag{19}$$

and its worst-case execution time is

$$T_{\text{adjac}} = m(6T_{\text{resp}} + 3T_p) + T_p + 2T_{\text{resp}} + m(6T_{\text{resp}} + 3T_p). \tag{20}$$

6. *Execution Time for Ordered Retrieval Algorithm*

The execution time for ordered retrieval depends on the number of memory words to be retrieved. The major processing step is a repeated search for an extremum (a search for a minimum for ascending order and a search for a maximum for descending order).

The best-case execution time would be obtained when there are no multiple matches (e.g., only a single bit set to one in the R register at the end of each extremum search operation). In such a situation, the contents of the R register are directly routed to the output unit where they are used to select a single word from the storage array. In case of multiple matches and depending on the application, we may need to select only a single word at a time. In this case, the output of R is routed to the response unit for selecting the first match, which will be indicated in register P. Then register P will be routed to the output unit for word retrieval. Thus the best-case execution time is

$$T_{\text{sort}} = n[m(6T_{\text{resp}} + 3T_p) + 2T_p + 2T_{\text{resp}}], \tag{21}$$

and the worst-case time is

$$T_{\text{sort}} = n[\overbrace{m(6T_{\text{resp}} + 3T_p)}^{1} + \overbrace{T_p}^{2} + \overbrace{\log_2 n(T_p + T_{\text{resp}})}^{3} + \overbrace{T_p}^{4} + \overbrace{3(T_{\text{resp}} + T_p)}^{5}]. \tag{22}$$

The number over the braces in the worst-case execution-time expression correspond to the following subtasks: (1) searching for an extremum, (2) routing register R to the response unit, (3) selecting the first responder in the priority register P, (4) routing P to the output unit, and (5) outputting the selected word. Note that the time it takes to route R to the selection unit and to disable the selected word in the ER

Table IV. Estimated Execution Time of the Parallel Algorithms on an Optical Content-Addressable Parallel Processor

Search Algorithm	Minimum Execution Time[a]	Maximum Execution Time[a]
Equivalence search	$3T_{resp} + 2T_p$	$3T_{resp} + 2T_p$
Threshold search	$4T_{resp} + 2T_p$	$m(6T_{resp} + 3T_p)$
Extremum search	$m(6T_{resp} + 3T_p)$	$m(6T_{resp} + 3T_p)$
Double-Limit search	$10T_{resp} + 5T_p + T_i$	$2m(6T_{resp} + 3T_p) + T_p + T_i + 2T_{resp}$
Adjacency search	$(m + 1)(6T_{resp} + 3T_p)$	$2m(T_{resp} + 3T_p) + T_p + T_{resp}$
Ordered retrieval	$n[m(6T_{resp} + 3T_p) + 2T_p + 2T_{resp}]$	$n[(6m + 3)T_{resp} + (3m + 5)T_p + \log_2 n(T_p + T_{resp})]$

[a]The parameters m and n represent the word length and the number of operands, respectively.

register is overlapped with that of routing R to the output unit and outputting the selected word.

Table 4 summarizes the estimated execution time for the algorithms presented. It is important to note that the execution time for the equivalence search, the threshold search (best-case), and the double-limit search (best-case) is a constant factor and is independent of the number of words in memory. The time for the threshold search (worst-case), the double-limit search (worst-case), the adjacency search, and the extremum search is proportional to the word length and is independent of the number of words involved in the operation. The delay time in ordered retrieval (sorting) is proportional to the product of the number of words to be sorted with the word length (best case). This advantage translates into a speedup factor of m (number of bits per word) more than that for electronic CAM's for the fundamental search algorithms. It is expected that such a speedup factor combined with the high speed at which these algorithms can be optically executed will result in a system throughput far better than any electronic associative machine can achieve.

VII. Discussions

Associative processing based on content-addressable memories has been argued to be the natural solution for nonnumerical information processing applications. Unfortunately, the implementation requirements of these architectures when one uses conventional electronic technology have been cost prohibitive; therefore associative processors have not been realized. Instead, software methods that emulate the behavior of associative processing have been promoted and mapped onto conventional location-addressable systems. However, this does not effect the natural parallelism of associative processing, namely, the ability to access many data words simultaneously. Optics has the advantage over electronics of directly supporting associative processing by its providing economic and efficient interconnects, massive parallelism, and high-speed processing.

This paper has presented the principles and initial design concepts of an associative architecture that matches well with optics advantages and is therefore highly amenable to optical implementation. The architecture relies heavily on the use of space-invariant interconnections, optical signal broadcasting and funneling (combining), and the simultaneous application of the same operation to many data points (single-instruction multiple data mode of computing). The motivation behind this architecture is to take advantage of the ease with which these operations can be realized with optics. A representative set of search algorithms have been presented to show the use and merits of the architecture. These algorithms are key components that occur in large computing tasks. It is important to note that these fundamental search algorithms are implemented on the optical architecture with an execution time independent of the problem size (the number of words to be processed). This indicates that the architecture would be best suited to applications in which the number of data sets to be operated on is high. We are currently conducting a comprehensive study to determine the type and range of applications for which the optical architecture is most suitable. Some of the applications being investigated are (1) real-time information retrieval, (2) database management, (3) knowledge-base and expert system implementation, (4) list and string processing, (5) bulk (numerical) processing (e.g., image processing), (6) pattern and speech recognition, and (7) implementation of data-driven architectures. At the architecture level, we are currently extending the one-dimensional matching concept (a single search word is compared with the two-dimensional array stored words) to a two-dimensional scheme by which several search words are simultaneously compared with the two-dimensional array. Such an extension will have a major impact on database and knowledge-base processing.

We have derived a list of requirements in order to optically implement the architecture, and we have presented a preliminary and simple version of an implementation that meets these requirements. This initial implementation version is meant to show only the feasibility of the architecture with existing optical nonlinear devices and conventional components. No optimization attempts were made. Nevertheless, this preliminary version reveals several key design issues that will determine the physical realization of such an optical architecture. Even if we assume the availability of optical nonlinear devices (latches and NOR gates)

in large sizes, the effective memory size will be critically determined by the beam spreading–combinling optics, the contrast ratio, and the fan-in and fan-out factors of the logic elements to be used.

This research was supported by National Science Foundation grant MIP-8909216. The author thanks the anonymous referees for their valuable suggestions.

References

1. K. Hwang and D. Degroot, *Parallel Processing for Supercomputers and Artificial Intelligence* (McGraw-Hill, New York, 1988).
2. R. Halstead, "Parallel symbolic computing," Computer **19**(8), 35–43 (1986).
3. T. Kohonen, *Content-Addressable Memories* (Springer-Verlag, New York, 1980).
4. C. J. Date, *An Introduction to Database Systems* (Addison-Wesley, Reading, Mass., 1986).
5. R. M. Lea, "VLSI and WSI associative string processors for cost-effective parallel processing," Computer J. **29,** 486–494 (1986).
6. C. C. Foster, *Content Addressable Parallel Processors* (Reinhold, New York, 1976).
7. R. M. Lea, "Information processing with an associative parallel processor," Computer **8**(11), 25–32 (1975).
8. C. Y. Lee and M. C. Paul, "A content addressable distributed logic memory with applications to information retrieval," Proc. IEEE **51,** 924–932 (1964).
9. K. Hwang and F. Briggs, *Computer Architectures and Parallel Processing* (McGraw-Hill, New York, 1984).
10. G. S. Almasi and A. Gottlieb, *Highly Parallel Computing* (Addison-Wesley, Reading, Mass., 1989).
11. A. A. Sawchuk and T. C. Stand, "Digital optical computing," Proc. IEEE **72,** 758–779 (1984).
12. A. Huang, "Architectural considerations involved in the design of an optical digital computer," Proc. IEEE **72,** 780–787 (1984).
13. W. T. Cathey, K. Wagner, and W. J. Miceli, "Digital computing with optics," Proc. IEEE **77,** 1558–1572 (1989).
14. A. Louri, "3-D optical architecture and data-parallel algorithms for massively parallel computing," IEEE Micro **11**(2), 24–68 (1991).
15. K. Hwang and A. Louri, "Optical multiplication and division using modified signed-digit symbolic substitution," Opt. Eng. **28,** 364–373 (1989).
16. B. K. Jenkins, P. Chavel, R. Forchheimer, A. A. Sawchuk, and T. C. Strand, "Architectural implications of a digital optical processor," Appl. Opt. **23,** 3465–3474 (1984).
17. Y. Li, D. H. Kim, A. Kostrzewski, and G. Eichmann, "Content-addressable memory-based optical modified signed-digit arithmetic," Opt. lett. **14,** 1254–1256 (1989).
18. F. Kiamilev, S. C. Esner, R. Paturi, Y. Fainman, P. Mercier, C. C. Guest, and S. H. Lee, "Programmable optoelectronic multiprocessors and their comparison with symbolic substitution for digital optical computing," Opt. Eng. **28,** 396–409 (1989).
19. P. B. Berra, A. Ghafoor, M. Guizani, S. J. Marcinkowski, and P. A. Mitkas, "Optics and supercomputing," Proc. IEEE **77,** 1797–1815 (1989).
20. P. B. Berra, K. H. Brenner, W. T. Cathey, H. J. Caulfield, S. H. Lee, and H. Szu, "Optical database/knowledgebase machines," Appl. Opt. **29,** 195–205 (1990).
21. A. D. McAulay, *Optical Computer Architectures: The Application of Optical Concepts to Next Generation Computers* (Wiley, New York, 1991).
22. K. Wagner and D. Psaltis, "Multilayer optical learning networks," Appl. Opt. **26,** 5067–5076 (1987).
23. H. J. Caulfield, J. Kinser, and S. K. Rogers, "Optical neural networks," Proc. IEEE **77,** 1573–1583 (1989).
24. A. Louri and K. Hwang, "A bit-plane architecture for optical computing with 2-d symbolic substitution algorithms, in *Proceedings of the 15th International Symposium on Computer Architecture* (Institute of Electrical and Electronics Engineers, New York, 1988).
25. C. Warde and A. Fisher, "Spatial light modulators: applications and functional capabilities," in *Optical Signal Processing,* J. Horner, ed. (Academic, New York, 1987), pp. 478–524.
26. J. A. Neff, R. A. Athale, and S. H. Lee, "Two-dimensional spatial light modulators: a tutorial," Proc. IEEE **78,** 836–855 (1990).
27. N. Streibl, K. H. Brenner, A. Huang, J. Jahns, J. Jewell, A. W. Lohmann, D. Miller, M. Murdocca, M. E. Prise, and T. Sizer, "Digital optics," Proc. IEEE **77,** 1954–1970 (1989).
28. D. A. B. Miller, D. S. Chemla, D. J. Eilenberger, P. W. Smith, A. C. Gossard, and W. T. Tsang, "Large room-temperature optical nonlinearity in $GaAs/Ga_{1-x}Al_xAs$ multiple quantum well structures," Appl. Phys. Lett. **44,** 821–823 (1982).
29. D. A. B. Miller, D. S. Chemla, T. C. Damen, A. C. Gossard, W. Wiegmann, T. H. Wood, and C. A. Burrus, "The quantum well self-electro-optic effect device: optoelectronic bistability and oscillation, and self-linearized modulation," IEEE J. Quantum Electron. **QE-21,** 1462–1476 (1985).
30. A. L. Lentine, H. S. Hinton, D. A. B. Miller, J. E. Henry, J. E. Cunningham, and L. M. F. Chirovsky, "Symmetric self-electrooptic effect device: optical set–reset latch, differential logic gate, and differential modulator/detector," IEEE J. Quantum Electron. **25,** 1928–1936 (1989).
31. S. D. Smith, J. G. H. Mathew, M. R. Taghizadeth, A. C. Walker, B. S. Wherret, and A. Hendry, "Room temperature, visible wavelength optical bistability in ZnSe interference filters," Opt. Commun. **51,** 357–362 (1984).
32. J. L. Jewell, M. C. Rushford, and H. M. Gibbs, "Use of a single nonlinear Fabry–Perot étalon as optical logic gate," Appl. Phys. Lett. **44,** 172–174 (1984).
33. G. R. Olbright, R. P. Bryan, K. Lear, T. M. Brennan, G. Poirier, Y. H. Lee, and J. L. Jewell, "Cascadable laser logic devices: Discrete integration of photoresistors will surface-emitting laser diodes," Electron. Lett. **27,** 216–218 (1991).
34. A. W. Lohmann, "What classical optics can do for the digital optical computer," Appl. Opt. **25,** 1543–1549 (1986).
35. A. A. Sawchuk and B. K. Jenkins, "Dynamic optical interconnections for optical processors," in *Optical Computing,* J. Neff, ed., Proc. Soc. Photo-Opt. Instrum. Eng. **625,** 145–153 (1986).
36. M. J. Murdocca, A. Huang, J. Jahns, and N. Streibl, "Optical design of programmable logic arrays," Appl. Opt. **27,** 1651–1660 (1988).
37. A. Hartmann and S. Redfield, "Design sketches for optical crossbar switches intended for large-scale parallel processing applications," Opt. Eng. **28,** 315–328 (1989).
38. J. Taboury, J. M. Wang, P. Chavel, F. Devos, and P. Garda, "Optical cellular logic architecture 1: Principles," Appl. Opt. **27,** 1643–1650 (1988).
39. M. Prise, N. C. Craft, R. E. LaMarche, M. M. Downs, S. J. Walker, L. Asaro, and L. M. F. Chirovsky, "Module for optical logic circuits using symmetric self-electrooptic effect devices," Appl. Opt. **29,** 2164–2170 (1990).
40. M. Prose, N. C. Craft, M. M. Downs, R. E. LaMarche, L. A.

Asaro, L. Chirovsky, and M. Murdocca, "Optical digital processor using arrays of symmetric self-electrooptic effect devices," Appl. Opt. **30,** 2287–2296 (1991).
41. A. L. Lentine, D. A. Miller, J. E. Henry, J. E. Cunningham, L. M. Chirovsky, and L. A. Asaro, "Optical logic using electrically connected quantum well PIN diode modulators and detectors," Appl. Opt. **29,** 2153–2163 (1990).
42. B. Hill, "The current status of two-dimensional spatial light modulators," in *Optical Computing: Digital and Symbolic,* R. Arrathoon, ed. (Dekker, New York, 1989), pp. 1–40.
43. R. A. Athale, "Optical matrix processors," in *Optical and Hybrid Computing,* H. H. Szu, ed., Proc. Soc. Photo-Opt. Instrum. Eng. **634,** 96–111 (1986).
44. A. A. Sawchuk, C. S. Raghavandra, B. K. Jenkins, and A. Varma, "Optical cross-bar networks," IEEE Computer **20**(6), 50–62 (1987).
45. G. Gheen, "Optical matrix–matrix multiplier," Appl. Opt. **29,** 886–887 (1990).
46. C. C. Foster, "Determination of priority in associative memories," IEEE Trans. Comput. **C-17,** 788–789 (1968).
47. M. J. Murdocca, *A Digital Design Methodology for Optical Computing* (MIT, Cambridge, Mass., 1990).